پوهنتون کاردان
KARDAN UNIVERSITY

Object Oriented Programming (Java)

Java Abstraction

Java Interfaces

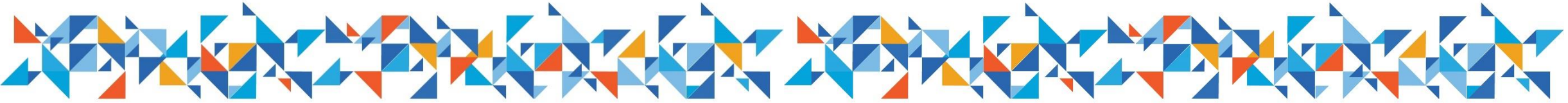
- Another way to achieve abstraction in Java, is with interfaces.
- An interface is a completely "abstract class" that is used to group related methods with empty bodies
- We can use java interfaces for unrelated classes.
- Java supports multiple inheritance using interfaces
- All the methods defined in interface are defined as public abstract

Interface Syntax

```
interface name{
//methods
}
```

Interface

```
public interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

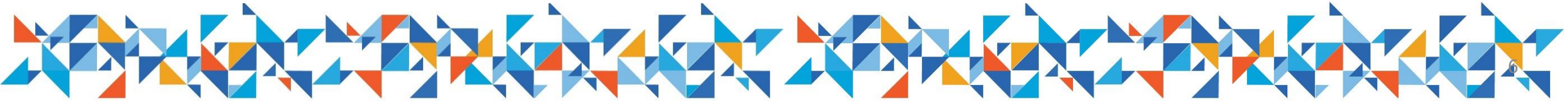


```
interface Bank2{
float rateOfInterest();
}
class AfghanInternational implements Bank2{
public float rateOfInterest(){
    return 9.15f;}
}
class Muslim implements Bank2{
public float rateOfInterest(){
    return 4.7f;}
}
class TestInterface{
public static void main(String[] args){
Bank2 A=new AfghanInternational();
System.out.println("Afghan International bank ROI: "+A.rateOfInterest()+"%");
Bank2 M=new Muslim();
System.out.println("Muslim bank ROI: "+M.rateOfInterest()+"%");

}}
```

```
interface Polygon {
  void getArea(int length, int breadth);
}
class Rectangle implements Polygon {
  public void getArea(int length, int breadth) {
 System.out.println("The area of the rectangle is "
          + (length * breadth));
  }}
class Testpol {
  public static void main(String[] args) {
    Rectangle r1 = new Rectangle();
    r1.getArea(5, 6);
  }
}
```

```
interface writer {
abstract void writing();
}
class notebook{
    public void writing(){
        System.out.println("In notebook we are writing");
    }
}
class pen extends notebook implements writer{
    public void writing(){
        System.out.println("By pen we can write anything in notebook");
    }
}
class interfacepractice{
    public static void main(String args[]){
        writer p=new pen();
        p.writing();
        notebook n=new notebook();
        n.writing();
    }
}
```

```java
interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
class Bicycle implements Vehicle{
    int speed;
    int gear;

    public void changeGear(int newGear){
        gear = newGear;
    }
     public void speedUp(int increment){
        speed = speed + increment;
    }
    public void applyBrakes(int decrement){
        speed = speed - decrement;
    }

    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }
}
```

```java
class Bike implements Vehicle {
    int speed;
    int gear;

    // to change gear
    public void changeGear(int newGear){
        gear = newGear;
    }
    public void speedUp(int increment){
        speed = speed + increment;
    }

    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }}
```

```java
class GFG {

    public static void main (String[] args) {

        Bicycle bicycle = new Bicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);

        System.out.println("Bicycle present state :");
        bicycle.printStates();

        Bike bike = new Bike();
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);

        System.out.println("Bike present state :");
        bike.printStates();
    }
}
```

How to use Interface

- To access the interface methods, the interface must be "implemented" by another class with the implements keyword (instead of extends).
- The body of the interface method is provided by the "implement" class:

```
class Bird implements Animal, ParentBird {
  public void animalSound() {
     System.out.println("The bird sounds");
  }
} public void sleep() {
     System.out.println(“Bird is sleeping");
}
```

Why And When To Use Interfaces?

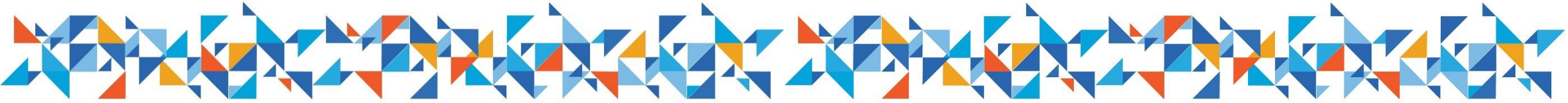
1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma.

```
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface,
SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}
```



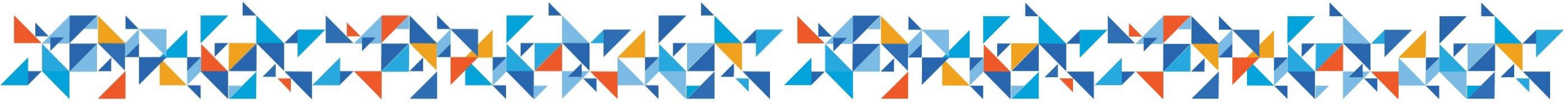
Example when to use Interface/Abstract

```
public interface LoginAuth{
        public String encryptPassword(String pass);
        public void checkDBforUser();
}
```

Suppose you have 3 databases in your application. Then each and every implementation for that database needs to define the above 2 methods:

```
public class DBMySQL implements LoginAuth{
        // Needs to implement both methods
}
public class DBOracle implements LoginAuth{
        // Needs to implement both methods
 }
public class DBAbc implements LoginAuth{
        // Needs to implement both methods
}
```

But what if encryptPassword() is not database dependent, and it's the same for each class? Then the above would not be a good approach.



Instead, consider this approach

```
public abstract class LoginAuth{
   public String encryptPassword(String pass){
          // Implement the same default behavior here
          // that is shared by all subclasses.
}
          // Each subclass needs to provide their own implementation of this only:
   public abstract void checkDBforUser();

 }
```

Now in each child class, we only need to implement one method - the method that is database dependent.

Abstract class vs Interface

- **Type of methods:** Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods.
- **Type of variables:** Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.
- **Implementation:** Abstract class can provide the implementation of interface. Interface can’t provide the implementation of abstract class.
- **Inheritance vs Abstraction:** A Java interface can be implemented using keyword “implements” and abstract class can be extended using keyword “extends”.
- **Multiple implementation:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
- **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

پوهنتون کاردان
KARDAN UNIVERSITY

Thank You…!